Implementing Swarm Production System with Multi-Robot Simulation

Avhad, Akshay; Arnarson, Halldor; Schou, Casper; Madsen, Ole

Published in:
Procedia Computer Science

DOI (link to publication from Publisher):
10.1016/j.procs.2024.01.093

Publication date:
2024

Document Version
Publisher's PDF, also known as Version of record

Citation for published version (APA):
Avhad, A., Arnarson, H., Schou, C., & Madsen, O. (2024). Implementing Swarm Production System with Multi-Robot Simulation. *Procedia Computer Science*, *232*, 934-945. https://doi.org/10.1016/j.procs.2024.01.093

Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 232 (2024) 934–945

Procedia Computer Science

www.elsevier.com/locate/procedia

5th International Conference on Industry 4.0 and Smart Manufacturing

Implementing Swarm Production System with Multi-Robot Simulation

Akshay Avhad[a,*], Halldor Arnarson[b], Casper Schou[a], Ole Madsen[a]

[a]*Department of Materials and Production, Aalborg University, Fibigerstræde 16, Aalborg East 9220, Denmark*
[b]*Department of Industrial Engineering, The Arctic University of Norway, Lodve Langesgate 2, Narvik 8514, Norway*

Abstract

Swarm Production is a novel paradigm promoting a high degree of flexibility in material flow and reconfigurability in shop floor layout, enabling high-mix production in manufacturing through the induction of autonomous robots. The conceptual description promises a leap within smart manufacturing and the Industry 4.0 domain. Implementing this theoretical concept becomes crucial to the self-organising production domain and gradually learning the relevant technological stack, optimisation methods, objective functions, and shop floor constraints needed for Swarm Production. This research paper addresses the challenges of the simulation of autonomous mobile robots in the production environment and concludes with an exemplified case simulation. A simulation testbed visually demonstrates the concept with planning, scheduling and control systems, providing their feasibility and reliability.

Keywords: Swarm Production; Industry 4.0; Self-organising; Autonomous Robots; Simulation;

1. Introduction

Manufacturing paradigms evolved beyond Dedicated Manufacturing Lines (DML), driven by the mass customisation demand within the same production facility addressed by concepts like Flexible Manufacturing Systems (FMS) and Reconfigurable Manufacturing (RMS) incorporating the philosophy of flexible routing of jobs and planar layouts of machines on the shop-floor[1]. In the era of Industry 4.0, personalised customisation is becoming a reality with the ability to dynamically generate variants of products in an order facilitated by digitalised enterprise infrastructure, enabling suppliers, retailers, and consumers to interact on a common platform. Autonomously reconfigurable paradigms like Self-organizing Manufacturing Network (SOMN)[2], Biological Manufacturing System (BMS)[3], Self-Organizing Flexible Manufacturing System (SoFMS)[4] and Line-less mobile assembly systems (LMAS)[5]

* Akshay Avhad
E-mail address: akshayra@mp.aau.dk

1877-0509
10.1016/j.procs.2024.01.093

based on the factory-level self-organisation of production units; has been proposed to improve flexibility in customisation capability of a factory.

Nomenclature

SPS	Swarm Production System
LMAS	Line-less Mobile Assembly System
MMS	Matrix-Structured Manufacturing
FLMS	Fluid Manufacturing System
RMS	Reconfigurable Manufacturing System
FMS	Flexible Manufacturing System
DML	Dedicated Manufacturing Lines
SOMN	Self-organisation Manufacturing Network
BMS	Biological Manufacturing Systems
SoFMS	Self-Organizing Flexible Manufacturing System
AMR	Autonomous Mobile Robot
MRTA	Multi-Robot Task Allocation
TR	Transport Robot
WR	Workstation Robot
TM	Topology Manager
SM	Swarm Manager
SiL	Software-in-the-Loop
ERP	Enterprise Resource Planning
MES	Manufacturing Execution System
DES	Discrete Event Simulation
AGV	Automated Guided Vehicle
PV	Product Variant
PI	Product Instance
SLAM	Simultaneous Localization and Mapping
ROS	Robot Operating System
RTOS	Realtime Operating System

Schou et al. in [6] proposes a Swarm Production System (SPS) with a distinct production planning and control philosophy, a paradigm based on dynamically movable workstations that can be spatially distributed on the shop floor with a flexible material flow that enables multi-variant production. The process workstations and product flow between them are handled by autonomous mobile robots (AMRs), eventually forming a heterogeneous fleet of Workstations Robots (WRs) and Transportation Robots (TRs) representing process agents and product carrier agents, respectively. The scope of SPS implementation includes planning and control activities within a manufacturing domain. Figure 1 shows a holistic view of SPS encompassing sub-systems within SPS across the enterprise cloud level, on-premise edge and the local shopfloor level and illustrates various production topologies an SPS can incorporate. The lifecycle of an SPS undergoes multiple reconfigurations during the planning phase enabled by AMRs, along with an optimal material flow depending on the input production order volume and mix. The SPS planning problem is addressed in [7] with the Topology Manager (TM), while the execution of WRs and TRs on the shop floor is handled by a Swarm Manager (SM). However, implementing the SM in orchestrating WRs and TRs remains an open research gap in SPS lifecycle demonstration.

The TM explained in [7] estimates the spatial topology (factory layout) for deployment on the shop floor in the form of Line, Matrix or Swarm topologies illustrated in Figure 1; and optimises the estimated topology of WRs based on the product mix and volume of the input order originating from ERP/MES. The SM generates a relevant execution plan based on the optimised topology from the TM and later allocates carrier and process tasks to individual TRs and WRs, respectively. Implementing an SPS is vital to demonstrate and validate the advantages it offers to factories of the future

concept. The orchestration of AMRs in industrial production is a more significant challenge than other service-based applications due to operational factors like predictability, robustness, safety and compliance standards. A simulation of SM is essential to assess the performance of identified topologies with metrics like makespan and throughput while improving and extending the knowledge base about the challenges in the stochastic SPS environment. The NP-hard SPS planning yields an NP-hard estimation of the execution plan for the SM relevant to the specific layout obtained in the TM.

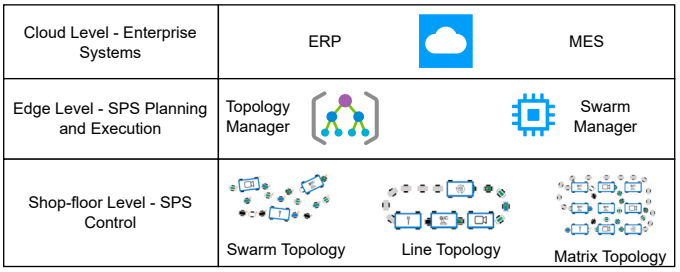

Fig. 1. Holistic architecture of SPS spanning planning and control Level showing the sub-systems involved in the operational life-cycle. ERP = Enterprise resource planning, MES = Manufacturing execution system.

The contributions of this paper are the software architectures based on the SM framework proposed in [8] and the Software-in-the-loop (SiL) simulation of SPS to validate the SM's architecture in the orchestration of WRs and TRs focussed on production missions. This research study also investigates suitable AMR simulation platforms in a production scenario, followed by a detailed explanation of the software architecture of the SM involved in the simulation of SPS and, finally, a validation of the architecture through an exemplified scenario.

2. Related Work

The use of AMRs on the shop-floor operations is promoted in the smart manufacturing paradigms that include SoFMS, BMS, SOMN, Matrix-Structured Manufacturing (MMS)[9], LMAS and Fluid Manufacturing Systems (FLMS)[10]. The strategic objectives of these paradigms exhibit a similarity; however, the classification could be achieved based on the planning, scheduling and execution methodology constituting the production lifecycle. A densely distributed network is anticipated in self-organizing networks; an intelligent orchestration is required for interoperability between machines, AMRs and tasks [11]. A priori statistical assessment in [12] emphasises estimating the NP-hard production planning problem in a self-organising LMAS. The potential applicability of these concepts depends on a visual exhibition, either with a simulation or a physical demonstrator adapted to the production environment.

In [13], FMS relies on a real-time rule-based scheduling algorithm that interacts with a Discrete Event Simulation (DES) environment to assess the production performance based on temporal and quantitative statistics. Simulations performed in [14] with an agent-based DES demonstrated the effectiveness of MMS over assembly line configuration with utilization and work package blockages of workstations for a set of orders. Control reference architecture proposed in [15] is validated through a DES simulation platform for real-time execution of an MMS demonstrating resource and job allocations to the AGVs and workstation cell. DES in [16] generates automated scenarios to generate training data for a neural network-based approximation model that predicts the performance of Line-less assembly system (LMAS) performances.

Multi-robot proprietary simulation testbed in [17] for the production line enables interdisciplinary learning and developed behavioural competencies with critical thinking towards problems in stochastic production environments. Various decentralized multi-robot task allocations are performed in a Gazebo[18] and Webots[19] environments to benchmark different scheduling approaches, with tasks associated with temporal, precedence and spatial constraints.

AMRs are primarily deployed in service-based applications, and most of these applications emulate pick-and-place tasks to the robots. The Fetch and Freight[20] AMR uses a Gazebo simulator to test a mobile manipulator that picks an object and navigates in a complex environment to place the object.

Open-source Robot Operating System (ROS) based simulation platforms like Gazebo and Webots imitate the physical and continuous time behaviour with explicit control over path planning, obstacle avoidance, and Simultaneous Localization and Mapping (SLAM) of manipulator AMR fleet. In comparison, DES is widely accepted for large-scale processes and material flow simulation in the manufacturing industry[21]. However, an SPS requires high-volume discrete product and task resource allocation and physics-enabled, multi-robot AMR, continuous time simulation interoperably to demonstrate and estimate the capability of large fleets of WRs and TRs. A commercially available platform like Visual Components [22] enables faster deployment with implicit fleet and local robot-level algorithms. It boasts an extensive library with profiles for most industrial robots and manufacturing machinery. A quantitive comparison is shown in Table 1 based on the capabilities of different simulation paradigms.

Table 1. Comparison of the capabilities in the context of SPS that different types of simulation engines provide

Capabilities	Discrete-Event Simulation	Physics-based Webots/Gazebo	Visual Components
Production KPIs	• Inherent Capability	• Requires development	• Inherent
Material Flow	• AGV/Conveyor based	• Open world flexibility	• Open world flexibility
AMR support	• Lacks support	• Extensive and explicit	• Extensive and implicit
Reconfiguration of production layout	• Faster discrete reconfiguration	• Slow physics enabled reconfiguration	• Faster and Adaptive reconfiguration
Multi-agent support	• Limited with API	• Open source with ROS	• OPC-UA enabled
Development time	• Faster Deployment	• Requires SLAM	• Faster without the need of SLAM

3. Implementation of SPS

The SPS life cycle illustrated in figure 2 commences with ERP & MES deployed on the cloud enterprise-level generating production orders that encapsulate product variants (PV), precedence flows for individual PV, and product instances (PI) required for each PV. Subsequently, the TM processes the input production order from ERP & MES, along with the workstation resource requirement to produce the order, and generates an initial logical topology without any spatial position. The TM is a graph-based and metaheuristic-based optimization layout planner for the SPS shopfloor structure. It shares a layout plan, including spatial locations for WRs after every reconfiguration requested by the SM during a changeover process invoked after substantial changes in the production order. The execution engine of SPS is the SM, which evaluates tasks based on the production order, allocates them to WRs and TRs during the runtime with a scheduling mechanism and manages the TR and WR fleet with traffic management and global positioning planning activities. All WR and TR agents are located at the control shop floor level and equipped with a working SLAM integrated with path planning and obstacle avoidance.

3.1. Swarm Manager Architecture

The SM description is based on the abstraction-specific C4[23] modelling, capable of complementing UML representations for detailed software architecture representations. The level of abstraction starts with an outermost `System-context`, explaining the interaction of the SM with other sub-systems based on Figure 2. A `Container` view in the middle focuses on the overview of SM software implementation. Finally, a combined `Component` and `Code` view describing the implementation of task planning, allocation and scheduling agents are shown in Figure 2.

3.1.1. System Context View

This outermost layer provides a vantage point for interconnected systems in the simulation of SPS illustrated in Figure 3. The SM is the execution control system for the multi-robot orchestration of TR and WR fleets, generating tasks based on the production order and relevant commands to the heterogeneous fleet of TRs and WRs on the shop floor. Ideally, the TM continuously monitors the production topology and computes the most optimal one. When the restructuring or changeover to optimised topology is cost-effective compared to deployed topology, a reconfiguration

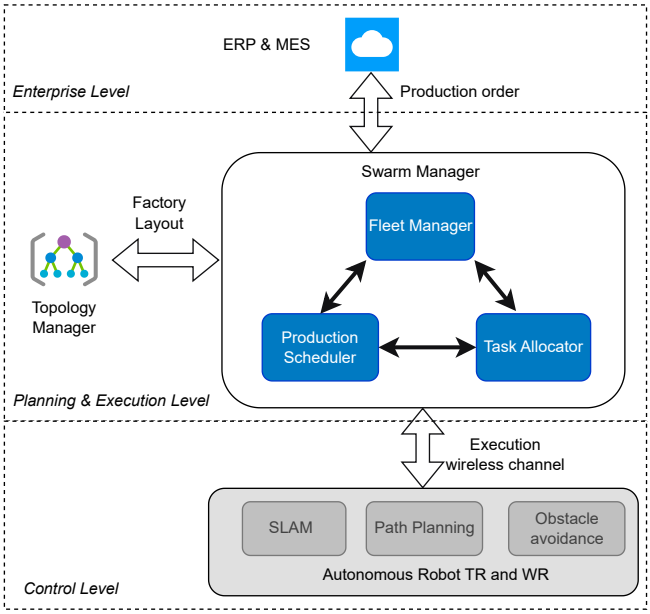

Fig. 2. Achitecture of SPS illustrating the information exchange during the life cycle. Enterprise level hosts a cloud-based ERP & MES, whereas the TM and SM handle the planning and execution. Control Level host robot agents TRs and WRs.

event is issued from TM. However, a viable option exists where TM optimises the SPS's layout at SM's reconfiguration request, usually occurring during a changeover of the input production order. The layout optimisation yields new WR positions that are transferred to the physical robots for reconfiguration, as shown in Figure 3

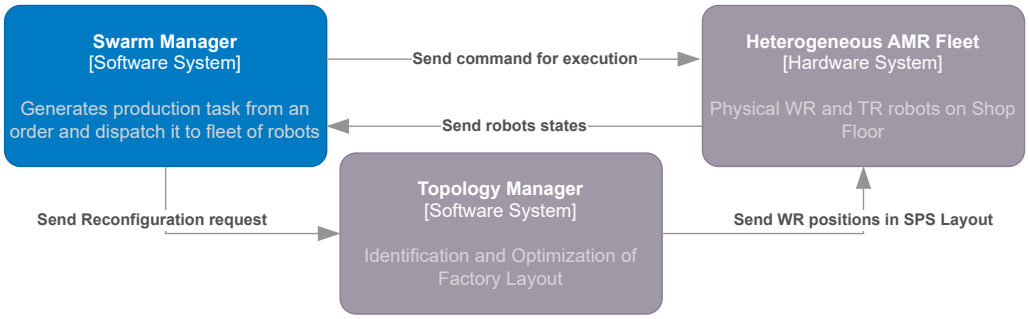

Fig. 3. System level viewpoint illustrating the interaction between multiple systems

3.1.2. Container Level

Figure 4 represents the C4 container layer focusing on the architecture within the SM implementation. The multi-agent SM system is developed in a `Python` programming environment with the asynchronous execution of TRs and WRs. The SM's execution flow commences with the generation of a PI, followed by relevant task generation for a TR for the PI and relevant execution command for the task to the TR agent. Multiple FIFO queues are embedded in separate threads to execute this sequential release of products, tasks and commands. Hence, complying with asynchronicity and concurrency in software engineering. The statistics during production runtime are stored in a NoSQL `mongoDB` database.

Main Thread: The main thread called `Manager` executes the planning, allocation, and scheduling functionalities to be explained in the section 3.1.3 for the Multi-Robot Task Allocation (MRTA) problem. The production initializes

with the creation of products based on the production order seen in Figure 5. Each product entity is an instance PI associated with a specific type of PV. Each PV has a distinct material flow represented as a sequence of tasks or precedence list of WRs defined in the production order data.

Product Release thread: The product release thread is a non-blocking I/O `AsyncIO` queue that receives the product object from the `Manager` and generates the production task based on the remaining task's list. The task required is sent back to the `Manager` for auctioning by broadcasting to every TR included in the simulation, and the best bid is rewarded with the task.

Task Waiting thread: If all the TRs are unavailable for tasks, e.g. due to being busy with the previously allotted task, the task remains unallocated. They are subsequently transferred to the `Task Waiting` thread, a non-blocking I/O `AsyncIO` queue to hold unallocated tasks during production runtime. Periodically (every 10 seconds), the tasks are released in the FIFO manner to the `Manager` for auctioning; the task is enqueued back to the thread in the event of being unallocated again.

Task Release thread: The task is a tuple comprising pickup and drop-off WR nodes. Once the task is successfully allocated, `Manager` assigns it to the `Task Release` thread's non-blocking I/O queue to initialise the task execution process. The thread transfers the queued task to be executed to the `Manager` for asynchronous path clearance check to ensure no other TR is blocking the path for task execution.

Command Release thread: After the task passes the clearance check in the `Manager`, it is ready for execution through the `Command Release` thread, which generates the command objects for physical robots. When required, the `Manager` sends product creation and reconfiguration commands to this thread. At the same time, it receives the encapsulated information of WRs and TRs locations, and their process states directly from the distributed robot controller.

3.1.3. Component Level

This section explains the component level in C4 modelling, elaborating on the `Manager` thread that hosts planning, allocation, TRs, WRs and scheduling software agents or components. Figure 5 represents the component level and the UML code supporting the abstract definition.

Task Planner agent: The input for the SM, i.e. `Order` shown in Figure 5 has a construct defined in the higher level MES & ERP. The `Order` data is processed to enlist PVs and PIs along with a precedence list of WR resources associated with every PV. After that, a global task array of $m \times n$ dimension, where m is the length of the precedence list, and n denotes the total number of PIs in `Order` data.

Scheduler agent: The execution starts with releasing products based on a `Greedy` approach during the initialisation phase. Tasks are generated for created products and sent to the Task Allocation agent for auctioning. The task during initialization includes generating the product at the `Source` Station followed by pickup and delivery task for TR agents, process execution at WR and concludes with delivery to `Sink` station. The transfer to `Sink` denotes the product's life-cycle completion in the SPS, requiring a new product to be generated at `Source` for an unfinished `Order`. During the normal production phase, i.e. after initialization and before sinking, the scheduler updates the product with the remaining tasks required for the product life-cycle at the end of task execution. The updated product object is relayed to the Product Release thread for task evaluation.

Task Allocation agent: The scheduler receives the task during initialization and later by the `Product Release` thread for auctioning. The auctioning process starts with broadcasting the tasks to the TRs in production. The TR agent responds with a bid value based on the marginal cost, accounting for the cost of attending to the task and the execution cost. The cost function is purely distance-based for the reactive scheduler in this simulation of SPS. The Task Allocation agent rewards the bid received through TR agents based on the minimum marginal cost in bid data. The task allocation for WRs is based on the process capability of the individual WR. For example, in a scenario where ev-

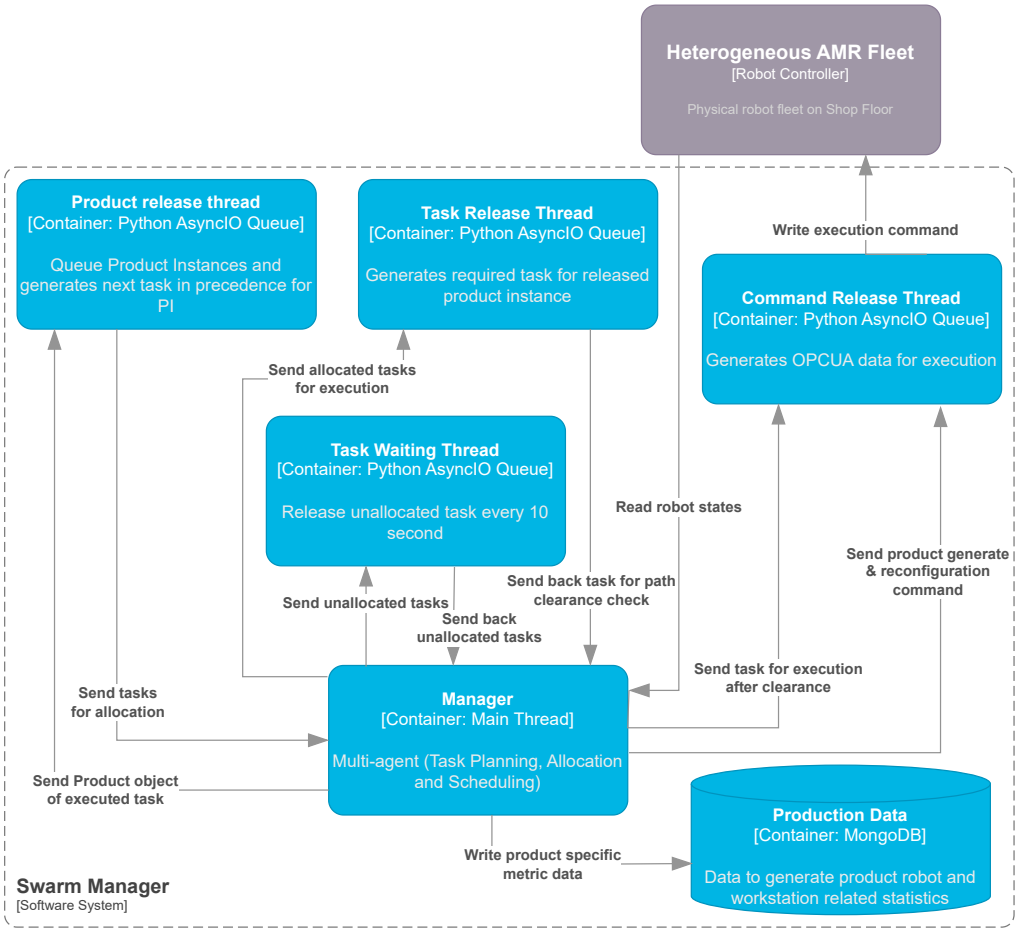

Fig. 4. Container level: Illustrating the architecture of the SM application.

ery WR has a unique ability to perform a process, the task allocation doesn't demand optimization and becomes direct.

AsyncIO EventLoop: The release of products and tasks is a concurrent and asynchronous process handled with queues across multiple threads. There are specific executions requiring asynchronicity and concurrency that include a path clearance check for TR pickup and delivery tasks, task execution time calculation for TR agents, and process execution on WRs. At the end of every process execution on the WRs, the scheduler is informed, followed by enqueuing PI objects based on the executed task to the `Product Release` thread. An `EventLoop`, the core of `AsyncIO`-based Python synchronicity, aggregates all the asynchronous functions and embeds them in the `Manager`.

3.2. Software-in-the-Loop architecture

The evaluation of SPS is performed with a SiL simulation that provides a testbed for the SM and the TM software prototype validation. The architecture of the SiL illustrated in Figure 6 includes the TM, the SM and a Visual Components platform to simulate the TRs and WRs. SM is extended to communicate with Visual components via the OPC-UA interface; both the OPC-UA interface and Visual Components are described in this section.

OPC-UA Interface: An OPC-UA server hosts the execution commands sent from the SM and robot states transferred by Visual Components. The data exchange between the SM and the OPC-UA server for Visual Components is managed on the OPC-UA client implementation within the SM. The SM sends commands for TR and WR tasks, PI

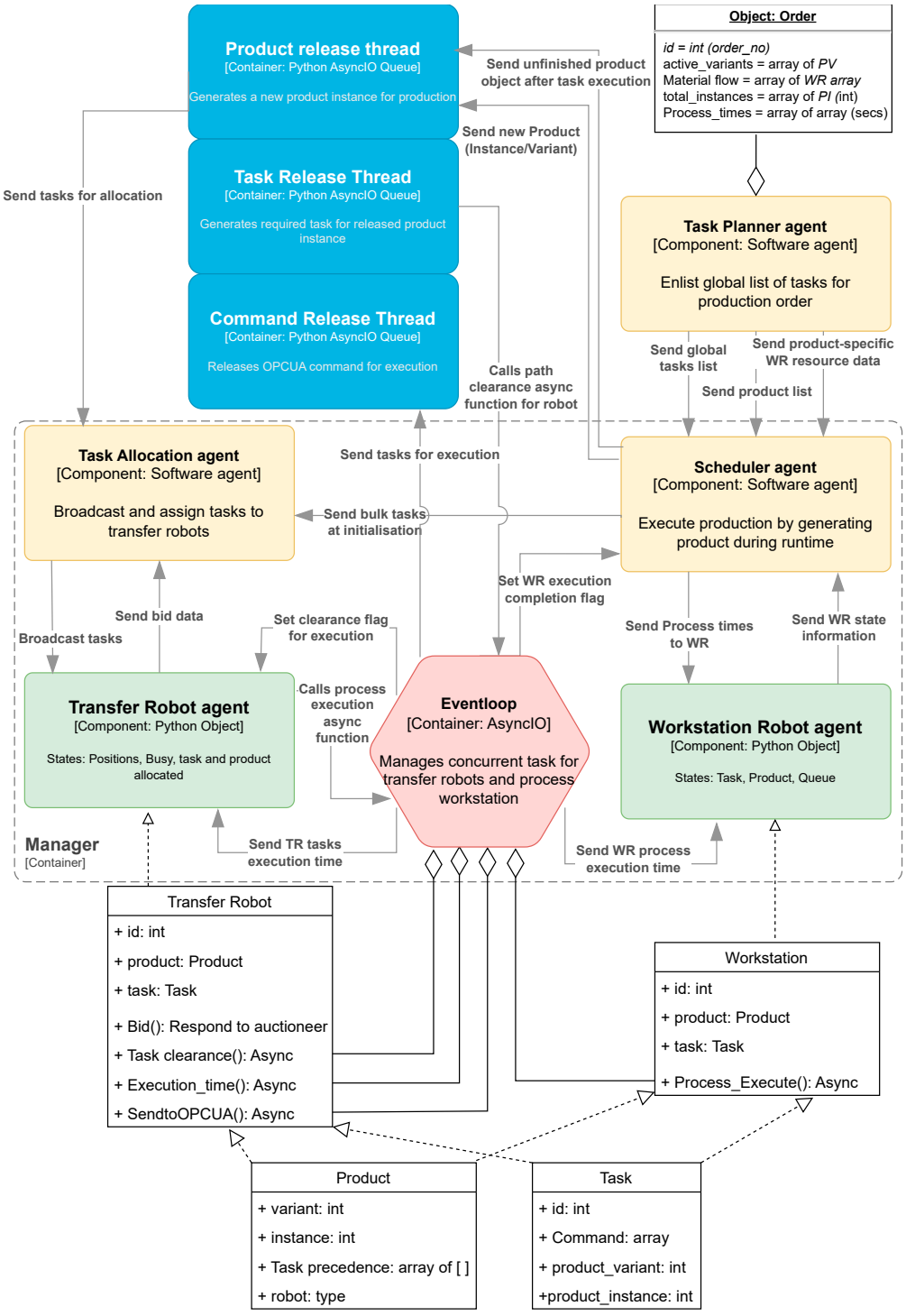

Fig. 5. Component Level: Illustrating the interaction of agents inside the Manager within SM.

generation and reconfiguration while it requests information on WR and TR states through the OPC-UA server.

Visual Components: The Visual Components framework features built-in functions for automated path planning and programming and a `Python` API for customizing simulations. However, out-of-the-box, Visual Components lacks a specific function for position control of mobile robots with path planning. To address this limitation, the robot controller for mobile robots has been modified to enable path planning with position control. Furthermore, additional functions for automated pick-and-place operations with robots have been integrated.

Visual Components also supports connectivity to an OPC-UA server as a client. This capability connects the software to the swarm manager using the OPC-UA standard. Through this connection, Visual Components facilitate the exchange of information regarding the position and status of AMRs, workstation locations, and the initiation of part creation and mission assignments for AMRs. Additionally, a function has been incorporated that allows the modification of workstation positions in the simulation through the OPC-UA server. Consequently, the system can test SPS scenarios where machines can be reconfigured for topology planning within the SPS lifecycle.

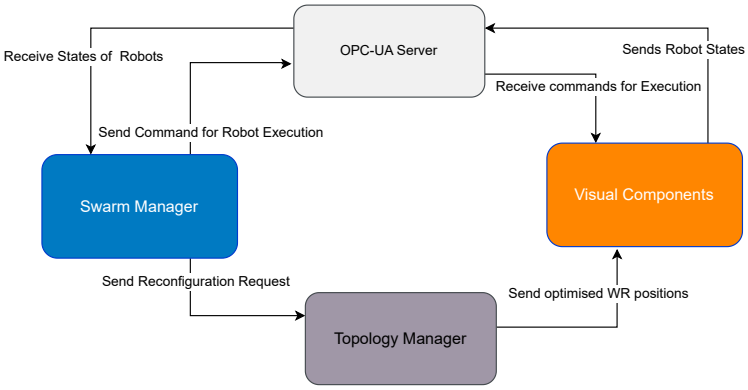

Fig. 6. Software-in-the-Loop architecture for Swarm Production Simulation

4. Exemplification

To exemplify the use of the SiL architecture with Visual Components, a proof-of-concept has been made with 3 TRs and 10 WRs inside the Visual Components environment as seen in Figure 7. The overall information flow has been described in Section 3.1.1. The demonstration starts with a simulation `Order` input to the SM and TM as illustrated in Figure 3. The TM generates near-optimal locations for WRs included in the `Order` data. The simulation demonstrates a `Greedy` approach in scheduling the execution of TRs. The production initialises with generating an unprocessed PI at Source equal to the TR fleet size (i.e. 3). The TR continues with the same PIs unless all the tasks required within each PI are executed and eventually transferred to `Sink`. A new PI is introduced until the last PI is produced and the total active PI is less than the total TRs. The implementation of the SM can be found in this repository [1]. Table 2 illustrates the production input order to the SM. A total of 5 products or PIs are included as test data, where each PI has a distinct material flow based on the PV it belongs to.

The application was tested on a Windows 11 operating system with AMD Ryzen 5900x, 32 GB RAM, and Nvidia RTX 2600S GPU. The objective function in SiL simulation is minimizing the tardiness (PI occupying a WR substantially longer than intended process times) and overall makespan. Although there are multiple optimization criteria in production scheduling, the initial prototype of SPS focuses on the solution's viability. Line, Matrix and Swarm topologies were used to assess the capability of the SM and functional testing of its software implementation, as shown in Figure 8. Individual product makespan is a sum of transfer time, i.e. cumulative time required during conveyance of a PI during production, followed by cumulative process time on all visited WRs. The product flow can experience

[1] https://github.com/zapcris/Swarm_Manager/tree/dadacdc7a28d7eadf6ec0622db35c0f143f67caf/Greedy_implementation

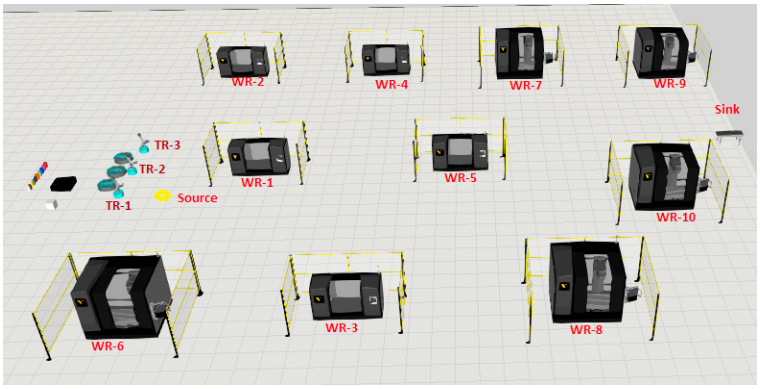

Fig. 7. Exhibiting matrix grid-shaped topology in SPS's factory environment in Visual Component 4.6.

Table 2. Illustrating test data for input order for the exemplified case. The Material Flow column represents a flow sequence through WRs, where 11 = `Source` & 12 = `Sink`. Process times are an array of time in seconds in the same order of WRs in Material Flow. The test order and 2D coordinates related to topology are included in the code repository.

PV	Material Flow	Total PI	Process times
1	[11, 1, 7, 12]	2	[4, 4]
2	[11, 2, 4, 6, 8, 12]	2	[10, 10, 10, 10]
3	[11, 3, 5, 6, 8, 9, 7, 12]	1	[10, 10, 10, 10, 10, 10]

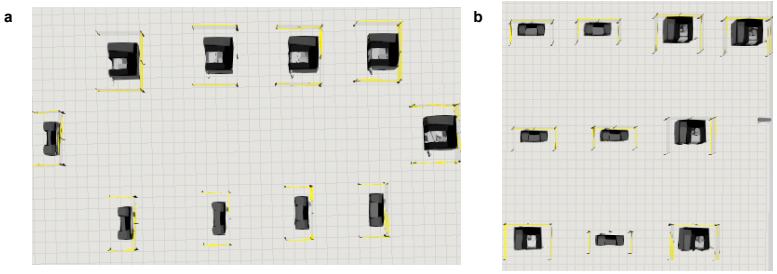

Fig. 8. (a) Linear Topology; (b) Swarm Topology.

blockages resulting specifically from a TR being unable to execute the carrier task due to the unavailability of WRs, as evident for P3 and P4 in all topologies in Figure 9. The high idle time in performance statistics in Figure 9 refers to inactivity or wait time when all the TRs are busy or on their way to serve. The nature of topology and the spatial positions of WRs impact the production makespan. The compact asymmetric swarm topology proves to be an efficient makespan compared to line and matrix topologies aligning with the hypothesis of SPS. The link to the simulation of the exemplified case with Visual Components 4.6 can be found here [2].

5. Conclusion

This research work yields the first demonstrator of an SPS based on SiL simulation. The SPS prototype exemplifies the proof-of-concept of the SM's multi-agent task planning, allocation and scheduling architecture. The performance chart in Figure 9 implies the SM's feasibility in assessing different topologies in the overall impact of the production

[2] https://www.youtube.com/watch?v=dlZ54ziZW80

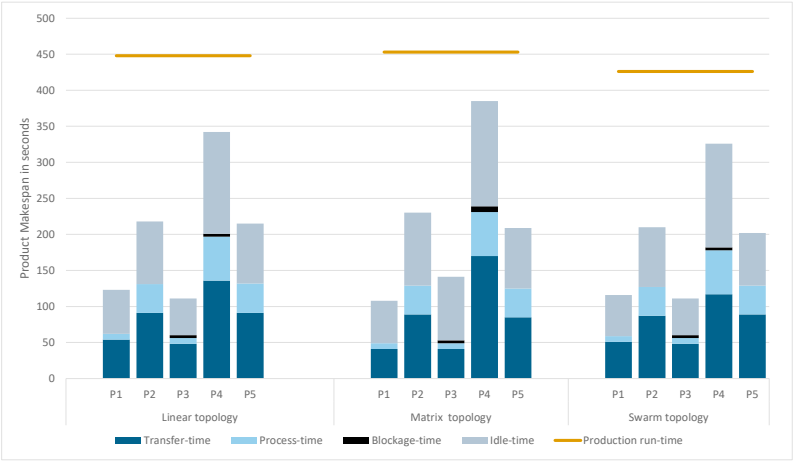

Fig. 9. Illustrating performance statistics of SiL simulation for different topologies based on the test input data in Table 2. The topology clusters, i.e. Linear, Matrix and Swarm on the x-axis encompass products P1 (PV=1 & PI=1), P2(PV=2 & PI=1), P3(PV=1 & PI=2), P4(PV=3 & PI=1), P5(PV=2 & PI=2)

lifecycle. The relation between product-level makespan and production cycle time seems stochastic, and optimising swarm topologies shall significantly impact the performance of the SPS. Even though the research aims to prove the viability of software architecture to simulate an SM, the performance metrics implemented in the prototype provide a vantage point for evaluating the efficiency of process and routing. Furthermore, they facilitate benchmarking various adaptive scheduling and optimal topology planning algorithms when implemented with an SM and a TM framework.

The software modelling in C4 provides layered abstractions incorporating multi-viewpoint, making the complex software implementation lucid, scalable and customisable. In a nutshell, SiL simulation enables the SM prototype to improve further in the ultimate objective of a real-world SPS execution engine. On the software level, `AsyncIO` provided an I/O level event-driven concurrency in the execution of task planning, allocation and scheduling along with asynchronous robot and workstation machine operation. AsyncIO-enabled concurrency is slower in execution as compared to a multi-processing architecture. However, combining asynchronous and multithreaded implementation eliminates the complex explicit sharing of information across the Manager, product, task and OPC UA-client threads. Thus enabling the necessary ad-hoc execution of tasks through PI objects and their relevant OPC-UA command release to TRs via multithreaded queues and managing the non-blocking I/O execution of TRs. The implementation in a Realtime Operating System (RTOS) can lead to a better computing resource allocation required for managing concurrent and parallel execution. Moving to a compiled language environment like `C++` or `Java` can enable faster execution than `Python's` interpreted type.

Visual Component provided an accessible and fast-to-build simulation environment for AMRs in a production scenario. The scalability of AMRs in Visual Components is subject to the capability of the computing system. Path planning, obstacle avoidance and global navigation of AMRs remain a black box. The SPS performance could be further enhanced with effective simulation with ROS and physics-based engines like Gazebo and Webots. However, a comprehensive simulation of a large-scale SPS scenario in physics-based simulation requires extensive effort.

The SM architecture exhibits an AMR fleet manager interoperable with the TM planner, production scheduler and order management in ERP/MES. Thus, the proposed SM architecture could also apply to production systems incorporating AMRs for inter-workstation logistics, especially in production paradigms like MMS, LMAS and FLMS.

Acknowledgements

This research has received funding from Innovation Fund Denmark within the Manufacturing Academy of Denmark (MADE FAST) programme and the European Union's Horizon 2020 research and innovation programme.

References

[1] Yoram Koren, Uwe Heisel, Francesco Jovane, Tosliiniichi Moriwaki, Guenter Pritschow, Galip Ulsoy, and Hendrik Van-Brussel. Reconfigurable manufacturing systems. *CIRP Annals*, 48(2):527–540, 1999.

[2] Zhaojun Qin and Yuqian Lu. Self-organizing manufacturing network: A paradigm towards smart manufacturing in mass personalization. *Journal of Manufacturing Systems*, 60:35–47, 2021.

[3] Kanji Ueda. A concept for bionic manufacturing systems based on dna-type information. In *Human Aspects in Computer Integrated Manufacturing*, pages 853–863. Elsevier, Amsterdam, 1992.

[4] Rotimi Ogunsakin, Nikolay Mehandjiev, and César A. Marín. Bee-inspired self-organizing flexible manufacturing system for mass personalization. In *From Animals to Animats 15*, pages 250–264, Cham, 2018. Springer International Publishing.

[5] Guido Hüttemann, Armin F. Buckhorst, and Robert H. Schmitt. Modelling and assessing line-less mobile assembly systems. *Procedia CIRP*, 81:724–729, 2019. 52nd CIRP Conference on Manufacturing Systems (CMS), Ljubljana, Slovenia, June 12-14, 2019.

[6] Casper Schou, Akshay Avhad, Simon Bøgh, and Ole Madsen. Towards the swarm production paradigm. In *Towards Sustainable Customization: Bridging Smart Products and Manufacturing Systems*, pages 105–112, Cham, 2022. Springer International Publishing.

[7] Akshay Avhad, Casper Schou, and Ole Madsen. Topology planning in swarm production system: Framework and optimization. In *Advances in Automotive Production Technology – Towards Software-Defined Manufacturing and Resilient Supply Chains*, pages 133–148, Cham, 2023. Springer International Publishing.

[8] Akshay Avhad, Casper Schou, and Ole Madsen. A framework for multi-robot control in execution of a swarm production system. *Computers in Industry*, 151:103981, 2023.

[9] Peter Greschke, Malte Schönemann, Sebastian Thiede, and Christoph Herrmann. Matrix structures for high volumes and flexibility in production systems. *Procedia CIRP*, 17:160–165, 2014. Variety Management in Manufacturing.

[10] Christian Fries, Manuel Fechter, Daniel Ranke, Michael Trierweiler, Anwar Al Assadi, Petra Foith-Förster, Hans-Hermann Wiendahl, and Thomas Bauernhansl. Fluid manufacturing systems (flms). In *Advances in Automotive Production Technology – Theory and Application*, pages 37–44, Berlin, Heidelberg, 2021. Springer Berlin Heidelberg.

[11] Jiaye Song, Zequn Zhang, Dunbing Tang, Haihua Zhu, Liping Wang, and Qingwei Nie. Designing and modeling of self-organizing manufacturing system in a digital twin shop floor. *International Journal of Advanced Manufacturing Technology*, pages 1–17, jan 2023.

[12] Robert H. Schmitt, Guido Hüttemann, and Sören Münker. A priori performance assessment of line-less mobile assembly systems. *CIRP Annals*, 70(1):389–392, 2021.

[13] Min Hee Kim and Yeong-Dae Kim. Simulation-based real-time scheduling in a flexible manufacturing system. *Journal of Manufacturing Systems*, 13(2):85–93, 1994.

[14] Malte Schönemann, Christoph Herrmann, Peter Greschke, and Sebastian Thiede. Simulation of matrix-structured manufacturing systems. *Journal of Manufacturing Systems*, 37:104–112, 2015.

[15] Christian P. Nielsen, Akshay Avhad, Casper Schou, and Elias Ribeiro da Silva. Control system architecture for matrix-structured manufacturing systems. *Computers in Industry*, 146:103851, 2023.

[16] Lea Grahn, Jonas Rachner, Amon Göppert, Sazvan Saeed, and Robert H. Schmitt. Framework for potential analysis by approximating line-less assembly systems with automl. In *Towards Sustainable Customization: Bridging Smart Products and Manufacturing Systems*, pages 423–430, Cham, 2022. Springer International Publishing.

[17] Shuting Wang, Liquan Jiang, Jie Meng, Yuanlong Xie, and Han Ding. Training for smart manufacturing using a mobile robot-based production line. *Frontiers of Mechanical Engineering*, 16:249–270, 2021.

[18] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. pages 2149 – 2154 vol.3, 04 2004.

[19] Olivier Michel. Cyberbotics ltd. webots™: professional mobile robot simulation. *International Journal of Advanced Robotic Systems*, 1(1):5, 2004.

[20] Melonee Wise, Michael Ferguson, Daniel King, Eric Diehr, and David Dymesich. Fetch & freight : Standard platforms for service robot applications. 2016.

[21] Chris J Turner and Wolfgang Garn. Next generation des simulation: A research agenda for human centric manufacturing systems. *Journal of Industrial Information Integration*, 28:100354, 2022.

[22] Visual Components - 3D manufacturing simulation software — visualcomponents.com. https://www.visualcomponents.com/.

[23] Andrea Vázquez-Ingelmo, Alicia García-Holgado, and Francisco J. García-Peñalvo. C4 model in a software engineering subject to ease the comprehension of uml and the software. In *2020 IEEE Global Engineering Education Conference (EDUCON)*, pages 919–924, 2020.